JT Harness

By Larry Hoffman and Jonathan Gibbons

November 2006

Copyright © 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Use is subject to license terms.

Sun, Sun Microsystems, Java, JavaTest and JavaHelp are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

L'utilisation est soumise aux termes de la Licence.

Sun, Sun Microsystems, Java, JavaTest et JavaHelp sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exlusivement par X/Open Company, Ltd.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la legislation americaine en matiere de controle des exportations et peuvent etre soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucleaires, des missiles, des armes biologiques et chimiques ou du nucleaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou reexportations vers des pays sous embargo des Etats-Unis, ou vers des entites figurant sur les listes d'exclusion d'exportation americaines, y compris, mais de maniere non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une facon directe ou indirecte, aux exportations des produits ou des services qui sont regi par la legislation americaine en matiere de controle des exportations et la liste de ressortissants specifiquement designes, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Please
Recycle

Adobe PostScript

Table of Contents

How the JT Harness Works 6
- Test Script 7
  - Two Types of Test Suites 7
- Test Finder 8
- Test Descriptions 8
  - Types of Test Descriptions 9
- Configuration Data 10
- Observers 11
- Remote Execution 12

JT Harness User Interface 12
- Graphical User Interface 12
  - Monitoring Test Status 13
  - Test Configuration 15
  - Analyze Test Results 16
  - Test Selection 17
  - Agent Monitor 17
- Command-Line Interface 18

Reports 18

Auditing Test Runs 18

JT Harness

The JT harness is the open source version of Sun's JavaTest™ harness commercial product.

The JT harness is a general purpose, fully-featured, flexible, and configurable test harness very well suited for most types of unit testing. Originally developed as a test harness to run TCK test suites, it has since evolved into a general purpose test platform upon which test suite developers can build product quality test suites that can run a wide variety of tests on a wide variety of platforms.

The JT harness is an excellent tool for configuring, sequencing, and running test suites that consist of large numbers of discrete, independent tests. It is especially good at testing APIs and compilers.

The JT harness user interface provides a convenient interface for configuring and running simple or extremely complex test suites. The user interface is continually being improved to make test configuration and execution simpler.

The JT harness is not a test development tool, however, it is shipped with a library of classes that make it easier to write tests. Although it does not help capture screen events to use in GUI tests, the JT harness can run tests created by such tools. It can also run tests that require human interaction.

The JT harness works best with test suites where the oracle for a test is included within the test.

This white paper explains how the JT harness works and describes many of the features of its user interface.

How the JT Harness Works

The JT harness is designed to provide flexible and customizable test execution, the tasks of controlling, finding, and executing the individual tests in a test suite are distributed among a number of different components:

JT harness UI-engine	The main JT harness program, the control center that provides support for scheduling, reporting, monitoring, analysis, and auditing. The JT harness user interface is described in "JT Harness User Interface" on page 12.
Test Script:	A Java™ platform class that the JT harness uses to execute the individual tests.
Test Finder	A Java platform class that finds and parses test descriptions for the test script.
Test Description	The set of parameters that are specified for each test. Typical entries include: the name of the test class file and data that effects how the test runs.
Configuration Data	Information about the computing environment in which the test suite is run. This information is used to convert logical steps into concrete actions. For example, the logical step: `compile foo.java` might be converted into the following concrete action: `C:\jdk1.3\bin\javac foo.java`
Observer APIs	The component that reports test events to both the GUI and command-line interfaces.

The following illustration shows the different JT harness components and their relationship to each other:

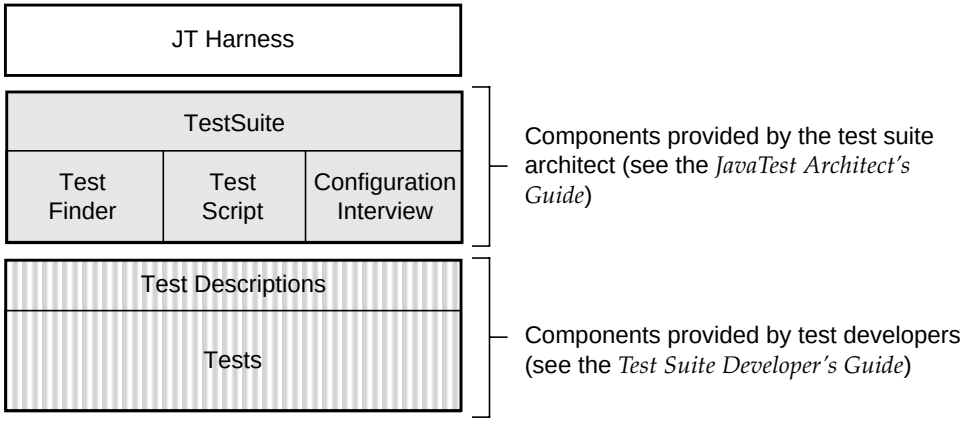

FIGURE 1 JT Harness Components

The next sections describe these components in more detail.

Test Script

The test script is responsible for running the tests and returning the status (pass, fail, error) to the JT harness. The test script must understand how to interpret the test description information returned to it by the test finder. The test script breaks down the execution of the test into a series of logical steps based on information from the test description. The test script can run the tests itself or delegate all or part of that responsibility to other commands. Test commands are Java platform classes that the test script instantiates to run tests. A fresh, new copy of the script is created for each test.

Tests can be executed either on the same computer on which the JT harness and the test script run, or the test script can delegate responsibility for remote execution to the JT agent. Remote execution is discussed in "Remote Execution" on page 12.

The JT harness provides a library of commands that test scripts can use to convert logical execution steps into concrete actions. It is also possible to create custom commands.

Two Types of Test Suites

Two types of test suites work well with the JT harness: *attribute based* and *procedure based*.

When test suite architects design a test suite for use with the JT harness, they have considerable flexibility about how to structure the tests. Attribute based and procedural based test suites offer different advantages and disadvantages:

Attribute based	Test execution logic is built into the test script (for example, the logical steps and their order). The test description consists of a list of attributes that supply specific parameters for each test that determine the sequence in which the steps are executed. This model has the effect of enforcing conformity within a test suite. The attribute model has proven very useful in conformance test suites like the JCK. FIGURE 2 on page 9 shows an HTML table that defines attributes in name-value pairs.
Procedural	In this model, the test script makes available a collection of execution steps that can be used by the test developer in any reasonable sequence. The test script knows what the possible actions are, but the test description defines the set of steps and their order for that particular test. This model is more flexible and has proven useful for more open-ended test suites used for regression testing. FIGURE 3 on page 10 shows an example of a procedural test description that uses Javadoc™ tool-style tagged comments at the beginning of the test code to define the test steps and their order.

Test Finder

The test finder is used by the test script to find, read, and parse test descriptions and pass the information back to the test script.

Test Descriptions

The test description is a set of parameters that are specified for each test, such as which file to execute or any other parameters that effect how the test is executed. Test descriptions are read and parsed by the test finder which then passes the parameters to the test script for use when executing the test. The following section describes the test descriptions used by the two standard test finders included with the JT harness.

Types of Test Descriptions

Test descriptions can be embodied in a large variety of formats. Essentially any format that can provide the necessary information to a test finder class works. Some examples:

HTML

The JCK test suite uses an HTML table to embody the test description for attribute based test suites. Following is an example of a simple JCK test description.

title	Checkbox Tests
source	CheckboxTests.java
executeClass	javasoft.sqe.tests.api.java.awt.Checkbox.CheckboxTests
executeArgs	-TestCaseID ALL
keywords	runtime positive

FIGURE 2 HTML-Based Test Description

The the benefit of this format is that it makes it easy and convenient for users to browse test descriptions using their web browsers. The trade-offs are more work for the test developers to create and maintain the HTML files, and the negative impact on performance caused by parsing these separate files.

In future releases of the JCK test suite, all the HTML test descriptions will be pre-compiled into a single, optimized format to improve performance and the uncompiled HTML descriptions will still be available for users to browse.

Tagged

Test descriptions can be embedded in comments in the test source code. The block of comments at the beginning of the following code fragment is an example of an embedded test description:

```
/* @test @(#)CheckActivateRef.java1.25 99/10/14
 * @bug 4105080
 * @summary Activation retry during a remote method call to an
 *          activatable object can cause infinite recursion in
 *          some situations.
 * @author John Brown
 *
 * @bug 4164971
 * @summary Allow non-public activatable class and/or
 *          constructor Main test class has a non-public
 *          constructor to ensure functionality is in place
 *
 * @library ../../../testlibrary
 * @build TestLibrary RMID
 * @build ActivateMe CheckActivateRef_Stub CheckActivateRef
 * @run main/othervm/policy=security.policy/timeout=240
*/

import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
     extends Activatable
     implements ActivateMe, Runnable
{

   private CheckActivateRef(ActivationID id, MarshalledObject obj)
     throws ActivationException, RemoteException
    {
     super(id, 0);
    }
[...]
```

FIGURE 3 Tagged Test Description

This format has the advantage of being very convenient for the test developer. The trade-off is that it is more difficult for the person running or debugging tests to view the test description.

Other formats: Test descriptions can also be embodied in text files, databases, XML, or any other format that can be found and parsed by a test finder class.

Configuration Data

Configuration data is the information required to convert logical test steps into concrete steps in the specific environment in which the tests are being run—for example, IP addresses, system names, and path names that change from system to system. The amount and type of configuration information required varies based on the test suite, and is influenced by the following factors:

- *Where and when* the configuration occurs. The architecture of the test suite determines where the configuration process occurs, specifically, whether configuration data is built directly into the test script and test description, or whether the tester must specify configuration information when the tests are run.
- The *amount* of configuration data. The amount of configuration information depends on the variety and complexity of the computing environments in which the tests run. If the tests always run on the same platform, configuration is *bounded* and very simple. If the tests can run in a wide, *unbounded* range of computing devices configuration is much more complicated:

  Bounded — When the set of computing environments in which tests are run is finite, all or most of the configuration can be built directly into the test script. In this case the test script understands the environment in which it runs (even if it is very complex), and the person running the test is not required to provide the information.

  Unbounded — When the set of computing environments is unbounded and and therefore probably unknown until the time the tests are run, the script must be very flexible. In this case, configuration cannot be built into the script and the person running the test is required to provide the information. The JCK is an example of this kind of test suite, it runs in a wide range of devices, from servers and desktop computers, to palm devices and cell phones.

  The JT harness provides a toolkit for creating configuration interviews to aid in complex configurations. Interviews gather configuration data by interviewing the person running the test. For more information about interviews, see "Test Configuration" on page 15.

Observers

The observer component monitors various aspects of test behavior and reports the behavior to the user interface component. The observers can report on both coarse and fine grain events and can watch the output being generated as tests execute. The observer API is available to test suite architects who would like to write programs that monitor test execution. An example of such a program might be one that monitors test runs and notifies the tester if a certain percentage of tests are failing.

Remote Execution

If the platform under test does not have sufficient resources to run the harness, the tests can be run remotely, using an agent to run the tests on the test platform and communicate with the JT harness. The JT harness provides a general purpose agent (JT agent), but test architects can also create custom agents.

The JT agent is a lightweight program compatible with JDK™ 1.1, and does not use the Java SE platform, or Swing classes. The JT agent uses a bidirectional serial connection to communicate between the test platform and the JT harness—it supports both the TCP/IP and RS-232 protocols. Other types of serial connections can be added through the JT harness API, for example, infrared, parallel, USB, firewire connections can be added and modelled on the existing serial system. If a test platform meets the following requirements, the JT agent is likely to work well:

- The device supports a communication layer that can last the duration of a test (couple of minutes)
- The agent code can be loaded into the device

If the test platform does not meet these requirements the JT harness API can be used to create a custom agent.

The JT agent can be used to test the Java Platform, Standard Edition (Java SE platform) and CDC-based technologies. The ME Framework is used to test CLDC-based technologies.

JT Harness User Interface

This section describes the features of both the JT harness graphical user interface and the command-line interface.

Graphical User Interface

The JT harness has a sophisticated graphical user interface (GUI) that enables testers to do the following:

- Monitor test status.
- Evaluate and analyze test results.
- Easily configure the test environment.
- Easily include and exclude tests from test runs.
- Generate and view reports about test run information.
- Use the online help system to get information about all aspects of the harness.

FIGURE 4 shows the main JT harness window.

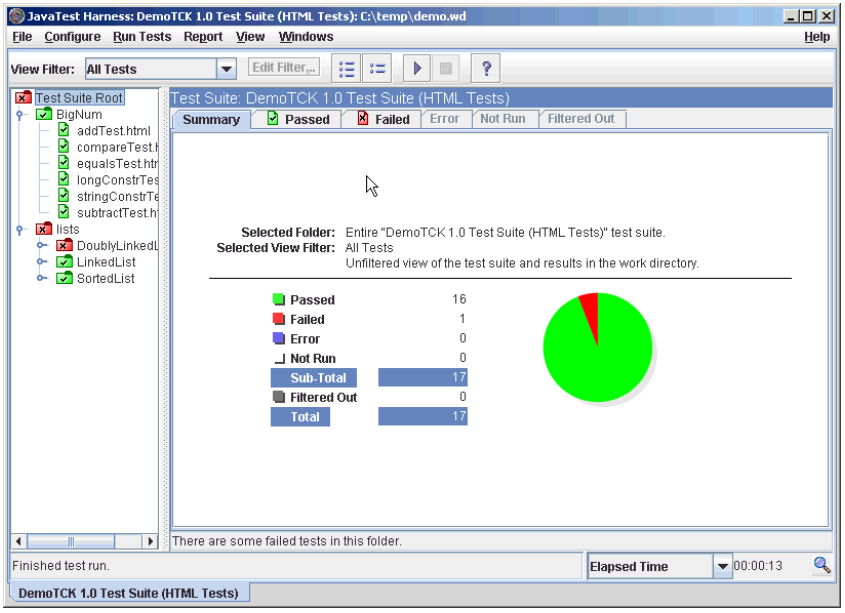
JavaTest Harness: DemoTCK 1.0 Test Suite (HTML Tests): C:\temp\demo.wd
File Configure Run Tests Report View Windows Help
View Filter: All Tests
Edit Filter...
Test Suite Root
BigNum
addTest.html
lists
DoublyLinkedL
LinkedList
SortedList
Test Suite: DemoTCK 1.0 Test Suite (HTML Tests)
Summary Passed Failed Error Not Run Filtered Out
Selected Folder: Entire "DemoTCK 1.0 Test Suite (HTML Tests)" test suite.
Selected View Filter: All Tests
Unfiltered view of the test suite and results in the work directory.
Passed 16
Failed 1
Error 0
Not Run 0
Sub-Total 17
Filtered Out 0
Total 17
There are some failed tests in this folder.
Finished test run.
Elapsed Time 00:00:13
DemoTCK 1.0 Test Suite (HTML Tests)

FIGURE 4 The JT Harness Main Window

Monitoring Test Status

The JT harness provides a number of ways to monitor the status of a test run.

Test Tree

The test tree pane shows the status of tests and test folders. The display is updated during the test run. FIGURE 5 shows a small test tree.

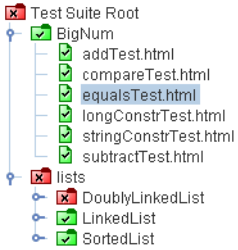

FIGURE 5 Test Tree

TABLE 1 shows the different icon colors and symbols and what they represent.

TABLE 1 Test Pane Icons and Descriptions

Icon	Result	Color and Description
	Error	A blue icon containing a ! symbol indicates that the test was not filtered out and that the JT harness could not execute it. These errors usually occur because the test environment is not properly configured.
	Failed	A red icon containing a X symbol indicates that the test was not filtered out and failed the last time it was executed.
	Not run	A white icon containing a – symbol indicates that the test was not filtered out but has not yet been executed.
	Passed	A green icon containing a ✓ symbol indicates that the test was not filtered out and passed the last time it was executed.
	Filtered out	A gray icon containing no symbols indicates that the test was currently not selected to be run.

Progress Monitor

For more detailed information about a test run, you can use the progress monitor window shown in FIGURE 6.

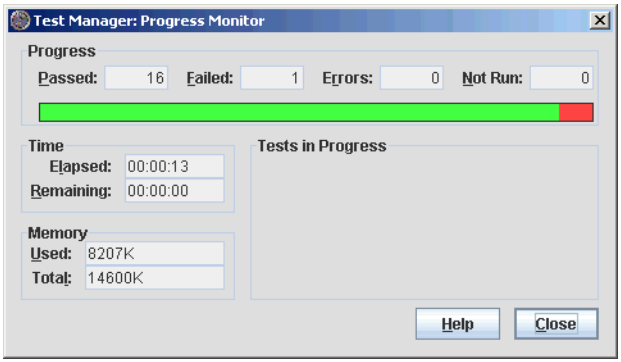

FIGURE 6 Progress Monitor Window

TABLE 2 describes the parts of the progress monitor window.

TABLE 2 Progress Monitor Description

Metric	Description
Progress	Shows the progress of the test run. The colors in the progress bar represent the classes of test results described in TABLE 1.
Time	The JT harness continually tracks the elapsed time of the current test run and estimates the time remaining to complete the test run.
Memory Usage	The JT harness tracks the amount of memory it uses during a test run and the amount of memory currently available.
Tests In Progress	During the test run, the JT harness displays the name of the test being read and executed.

Test Configuration

Because some test suites can be run in a virtually unbounded range of computing environments, configuring test environments can be a complicated task.

The JT harness Configuration Editor provides support for creating wizard-based configuration interviews that hide much of the complexity of unbounded configuration from the tester. Using a configuration interview, the tester only answers questions about the platforms involved and is not required to know anything about the syntax or format required for specifying the information to the JT harness. In addition, the Configuration Editor contains a help pane that provides examples and information about each question. At the completion of the interview, the configuration data can be exported to a format that the JT harness can read. The Configuration Editor supports the creation of configuration templates that make it

possible for a site administrator to fill in the configuration information common to a given company or site. This can greatly reduce the amount of configuration each user is responsible for.

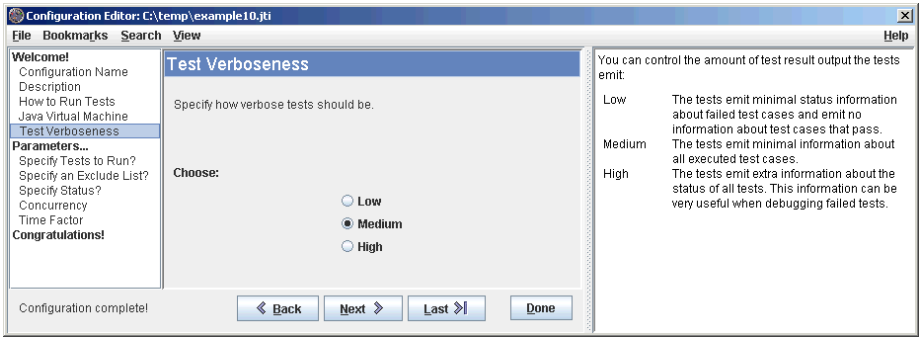
Configuration Editor: C:\temp\example10.jti
File Bookmarks Search View Help
Welcome!
Configuration Name
Description
How to Run Tests
Java Virtual Machine
Test Verboseness
Parameters...
Specify Tests to Run?
Specify an Exclude List?
Specify Status?
Concurrency
Time Factor
Congratulations!
Test Verboseness
Specify how verbose tests should be.
Choose:
Low
Medium
High
You can control the amount of test result output the tests emit:
Low The tests emit minimal status information about failed test cases and emit no information about test cases that pass.
Medium The tests emit minimal information about all executed test cases.
High The tests emit extra information about the status of all tests. This information can be very useful when debugging failed tests.
Configuration complete!
Back
Next
Last
Done

FIGURE 7 Configuration Editor

Test suite architects create the questions and help that comprise the interview which is displayed by the Configuration Editor.

Analyze Test Results

After the test run is complete the tester can browse the results of the run. By selecting a test in the display, the tester can view detailed information about the test including:

- Test description
- Test source files
- Configuration information that applies to that test
- Test run details
- Any output (messages) the test may have generated

This information can be used to debug and troubleshoot any execution problems.

FIGURE 8 shows the Test Messages pane. This pane is used to display output from the test and test script.

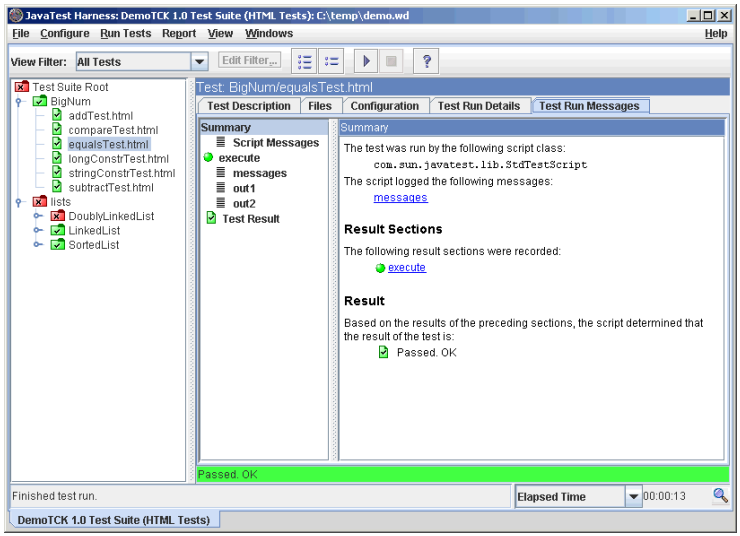

FIGURE 8 Test Messages Pane

Test Selection

The JT harness provides sophisticated methods for including and excluding specified tests from a given test run. Tests can be included and excluded from a test run based on the following:

- Keywords defined in the test description.
- Status from prior test runs (passed, failed, error, not run).
- Inclusion in a special *exclude file.*
- Tests selected by the tester. The tester can choose to run individual tests or if the test suite is arranged hierarchically, the tester can choose branches of the suite.
- Filters. The JT harness GUI enables you to select and create view filters that permit you to select which tests appear in the test tree.

Agent Monitor

If tests are executed remotely (see "Remote Execution" on page 12) the tester can monitor and coordinate the activity of agents using the agent monitor.

Command-Line Interface

The JT harness can also be invoked through its command-line interface. Using the JT harness command-line interface, it is possible to run tests in build scripts and other automated process. Almost all test execution functionality is available through the command-line interface.

Reports

You can generate detailed HTML summary reports after each test run. These reports can be browsed and printed using a web browser. You can generate and view reports about test run information, such as:

- Tests grouped by test status
- Configuration interview questions and answers
- Test environment used for the test run

Auditing Test Runs

The JT harness contains a tool that audits test runs. The audit tool determines if all required tests ran successfully and also verifies the integrity of the results files produced during the test run to ensure that they were not altered. At the completion of the audit process, a report is written.